DEFEATING THE TRAINING-INFERENCE MISMATCH VIA FP16

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) fine-tuning of large language models (LLMs) often suffers from instability due to the numerical mismatch between the training and inference policies. While prior work has attempted to mitigate this issue through algorithmic corrections or engineering alignments, we show that floating point precision is a major and controllable contributing factor. The widely adopted BF16, despite its large dynamic range, introduces large rounding errors that break the consistency between training and inference. In this work, we demonstrate that simply reverting to **FP16** substantially reduces this mismatch. The change is simple, fully supported by modern frameworks with only a few lines of code change, and requires no modification to the model architecture or learning algorithm. Our results suggest that using FP16 uniformly yields more stable optimization, faster convergence, and stronger performance across diverse tasks, algorithms and frameworks. We hope these findings motivate a broader reconsideration of precision trade-offs in RL fine-tuning.

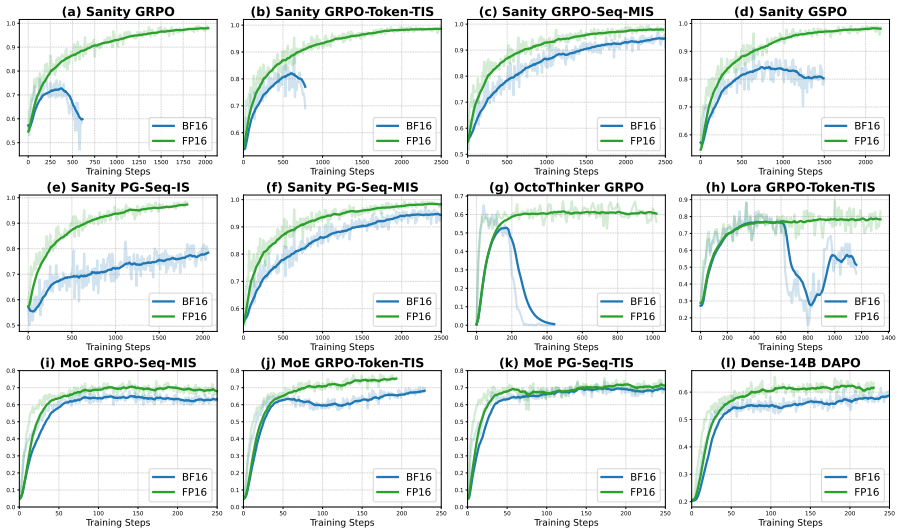

Figure 1: Training reward comparison between BF16 and FP16. We evaluate across diverse settings: our `Sanity` test (Section 4) with various algorithms (GRPO, GSPO, TIS, MIS, PG); different model families (R1D, Qwen and `OctoThinker`); alternative fine-tuning methods (`Lora`); and larger scale models (`Dense-14B`, `MoE`). Results are validated on two independent frameworks (VeRL and Oat).

1 INTRODUCTION

Reinforcement learning (RL) has emerged as a powerful paradigm for fine-tuning large language models (LLMs) to boost the reasoning performance (Guo et al., 2025; Zeng et al., 2025; Liu et al., 2025c; Qi et al., 2025). However, the path to achieve high-performing models through RL is often fraught with instability. The training process is notoriously sensitive to hyperparameters and can suffer from training collapse, making it a significant challenge to reliably improve model performance (Yu et al., 2025; Cui et al., 2025; Yao et al., 2025). This fragility has spurred a continuous search for

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

methods that can stabilize and streamline the RL fine-tuning process (Liu et al., 2025a; Team et al., 2025; Zheng et al., 2025a; Ma et al., 2025).

A critical source of this instability stems from a fundamental discrepancy in modern RL frameworks: the **training-inference mismatch**. To accelerate training, these frameworks typically use different computational engines, a highly optimized one for fast inference (rollout) and another for training (gradient computation). While mathematically identical, these engines produce numerically different outputs due to precision errors and hardware-specific optimizations. As recent work has highlighted (Yao et al., 2025; Liu et al., 2025a; Team et al., 2025; Zheng et al., 2025a), this seemingly minor mismatch between the inference and the training introduces significant issues into the optimization process.

Existing solutions have attempted to address this mismatch through algorithmic patches based on importance sampling. Notably, Yao et al. (2025) introduced a token-level importance sampling ratio as a patch to the GRPO (Shao et al., 2024) gradient. While this simple correction can prolong training, it was later shown by Liu et al. (2025a) to be insufficient to fully stabilize training due to its biased gradient. As an alternative, they proposed using a sequence-level importance sampling ratio for the correction. Although this method is more stable, its effectiveness is hampered by slow convergence speed, a direct consequence of the high variance inherent in sequence-level ratios. More fundamentally, both approaches only compensate for the biased gradient during training, while the underlying numerical mismatch between the engines persists. In principle, optimizing with one engine while deploying with another can create a **deployment gap**, where the resulting model is not truly optimal for its inference environment. This calls for a solution that eliminates the mismatch at its source, rather than merely compensating for it.

In this work, we take a step back from the complex algorithmic fixes and investigate a major source of the numerical mismatch: **floating-point precision**. We identify that the modern standard for mixed-precision training, BFloat16 (BF16), is the primary culprit. While BF16 has a wide dynamic range which is excellent for stable pre-training, its low precision makes it highly susceptible to rounding errors that accumulate and eventually cause the training and inference policies to diverge.

Our key finding is simple: switching from **BF16 to FP16** during RL fine-tuning substantially reduces the training–inference mismatch in our experiments. With more mantissa bits, FP16 offers higher numerical precision and makes results less sensitive to implementation differences between training and inference. This change can remove the need for complex algorithmic workarounds in the tested settings, making the purest importance-weighted policy-gradient estimator practical. Across our evaluated models, tasks, algorithms, and frameworks, FP16 training improves stability and often accelerates convergence, presenting a clean and efficient solution to a critical challenge in RL-based LLM alignment.

2 Background

In modern RL frameworks for LLM fine-tuning, different engines are used for inference and training to maximize system efficiency, which inevitably creates a mismatch between the inference policy $\mu(\cdot|\theta)$ and training policy $\pi(\cdot|\theta)$ due to subtle numerical discrepancies, even though, in principle, the two should be mathematically identical ($\mu = \pi$). This mismatch brings two issues elaborated below.

Biased Gradient To optimize the trainer policy $\pi(\cdot|\theta)$, we typically adopt the following objective:

$$\mathcal{J}(\theta) = \mathbb{E}_{x\sim p_{\mathcal{X}}}\Big[\mathcal{J}(x,\theta)\Big] = \mathbb{E}_{x\sim p_{\mathcal{X}}}\Big[\mathbb{E}_{y\sim\pi(\cdot|x,\theta)}[R_y]\Big], \tag{1}$$

where x is the prompt sampled from a distribution $p_{\mathcal{X}}$, y is the response, and R_y is the reward of y. We omit the dependency on x in the reward notation for brevity. The policy gradient can be calculated by REINFORCE estimator (Williams, 1992; Sutton & Barto, 2018):

$$\nabla_\theta\mathcal{J}(\theta) = \mathbb{E}_{x\sim p_{\mathcal{X}}}\Big[\nabla_\theta\mathcal{J}(x,\theta)\Big], \quad \text{where } \nabla_\theta\mathcal{J}(x,\theta) = \mathbb{E}_{y\sim\pi(\cdot|x,\theta)}\Big[\nabla_\theta\log\pi(y|x,\theta)\cdot R_y\Big].$$

In practice, we sample the responses from the inference policy μ, instead of the training policy π. As noted by Yao et al. (2025) and Liu et al. (2025a), the policy gradient would become biased if simply ignoring this mismatch.

$$\nabla_\theta\mathcal{J}_{\text{biased}}(x,\theta) = \mathbb{E}_{y\sim\mu(\cdot|x,\theta)}\Big[\nabla_\theta\log\pi(y|x,\theta)\cdot R_y\Big] \neq \nabla_\theta\mathcal{J}(x,\theta)$$

Deployment Gap Another important but hard to fix issue is the deployment gap. Though it is $\pi(\cdot|\theta)$ that we train, it is $\mu(\cdot|\theta)$ that we use for deployment and evaluation. However, the parameter θ optimized under the training engine π is not necessarily optimal for the inference engine μ:

$$\arg\max_\theta \mathbb{E}_{y\sim\mu(\cdot|x,\theta)}[R_y] \neq \arg\max_\theta \mathbb{E}_{y\sim\pi(\cdot|x,\theta)}[R_y].$$

This mismatch can make the optimizer and deployment target different objectives. While algorithmic patches (Yao et al., 2025; Liu et al., 2025a) address gradient bias, they do not by themselves make the two computations identical. This calls for a fundamental solution to reduce the mismatch.

2.1 Gradient Correction via Importance Sampling

To correct the biased gradient introduced by the training-inference mismatch, a principled approach is to use importance sampling (IS). This method re-weights the gradient calculation using a sequence-level probability ratio, ensuring the gradient estimator remains unbiased (see Appendix C.1 for trading a small amount of bias for variance reduction). The policy gradient for a given prompt x is thus corrected as:

$$\nabla_\theta \mathcal{J}_{\text{pg-seq-is}}(x) = \mathbb{E}_{y\sim\mu(\cdot|x,\theta')}\left[\rho \cdot \nabla_\theta \log \pi(y|x,\theta) A_y\right], \text{where } \rho = \frac{\pi(y|x,\theta)}{\mu(y|x,\theta')}, \tag{2}$$

where θ' denotes the parameters used for sampling, which may differ from θ in an off-policy setting. The term $A_y = R_y - B(x)$ is the advantage, with $B(x)$ serving as a baseline for variance reduction (Sutton & Barto, 2018).

2.1.1 Existing Implementations

Although inspired by the importance sampling principle, recent methods (Yao et al., 2025; Liu et al., 2025a) are effectively implemented as auxiliary patches on top of GRPO, rather than adhering to the strictly principled formulation.

The standard GRPO gradient (Shao et al., 2024), which does not correct for the training-inference mismatch, is calculated as follows (corrected by Dr.GRPO (Liu et al., 2025c)):

$$\begin{aligned} \nabla_\theta \mathcal{J}_{\text{grpo}}(x) &= \mathbb{E}_{y\sim\mu(\cdot|x,\theta')}\left[\sum_{t=1}^{|y|} \nabla_\theta \hat{r}_t(\theta)\right], \\ \hat{r}_t(\theta) &= \min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t\right), \quad r_t = \frac{\pi(y_t|x,y_{<t},\theta)}{\pi(y_t|x,y_{<t},\theta')}. \end{aligned} \tag{3}$$

For each prompt x, GRPO samples a group of G responses $\{y_i\}_{i=1}^G$ from the inference policy $\mu(\cdot|x,\theta')$ to compute the sequence-level advantage $A_y = R_y - \frac{1}{G}\sum_{i=1}^G R_{y_i}$, which is broadcast to every token in y.

Based on GRPO, Yao et al. (2025) introduced a token-level TIS correction:

$$\nabla_\theta \mathcal{J}_{\text{grpo-tok-tis}}(x) = \mathbb{E}_{y\sim\mu(\cdot|x,\theta')}\left[\sum_{t=1}^{|y|} \min\left(\frac{\pi(y_t|x,y_{<t},\theta')}{\mu(y_t|x,y_{<t},\theta')}, C\right) \cdot \nabla_\theta \hat{r}_t(\theta)\right], \tag{4}$$

Subsequently, Liu et al. (2025a) advanced this approach by proposing a sequence-level MIS variant, which uses a single ratio to determine whether the update is applied to the whole sequence:

$$\nabla_\theta \mathcal{J}_{\text{grpo-seq-mis}}(x) = \mathbb{E}_{y\sim\mu(\cdot|x,\theta')}\left[\mathbb{I}\{\frac{\pi(y|x,\theta')}{\mu(y|x,\theta')} \leq C\} \cdot \frac{\pi(y|x,\theta')}{\mu(y|x,\theta')} \cdot \sum_{t=1}^{|y|} \nabla_\theta \hat{r}_t(\theta)\right], \tag{5}$$

Compared to the vanilla policy gradient estimators (Equation (2) and its TIS/MIS variants), existing GRPO-based implementations require an additional forward pass to compute $\pi(\cdot|\theta')$ for their off-policy correction. This extra step incurs approximately 25% computational overhead during training, assuming a backward pass is twice as costly as a forward pass (Qi et al., 2023).

2.2 Engineering Attempts to Reduce the Mismatch

Another line of work attempts to mitigate the training-inference mismatch from an engineering perspective, but with limited success. Early attempts, such as using an FP32 language model head by Chen et al. (2025), are shown to be insufficient to prevent training collapse (Yao et al., 2025; Liu et al., 2025a). Later, Team et al. (2025) reported promising results by manually aligning training and inference implementations. However, this approach requires deep domain knowledge and substantial engineering effort, and it is unclear whether such bespoke fixes can be generalized across different frameworks or models. Very recently, approaches that seek to achieve bit-wise consistency between specific training and inference engines have emerged, by using specialized batch-invariant kernels to defeat non-determinism (He, 2025). While theoretically sound, this method faces practical challenges: it comes at a significant efficiency cost and leads to a brittle solution dependent on specific model architectures and kernels. This inflexibility poses a major barrier to adoption and maintenance, especially in complex, large-scale training scenarios that employ varied parallelization techniques.

Despite these efforts, the mismatch generally persists in practice due to fundamental differences between training and inference computations that are difficult to reconcile. For example, tokens are generated auto-regressively during inference but are processed in parallel during training. Different parallelization strategies and precision-sensitive operations such as top-k expert selection in Mixture-of-Experts (MoE) models further complicate the situation. This inherent difficulty highlights the need for a more fundamental solution that avoids such complex and brittle engineering workarounds.

3 Revisiting FP16 Precision

Table 1: Comparison of 16-bit Floating-Point Formats.

Property	**FP16**	**BF16**
Bit Allocation		
Exponent Bits	5	8
Mantissa Bits	10	7
Dynamic Range		
Smallest Positive Normal	$\approx 6.1 \times 10^{-\mathbf{5}}$	$\approx 1.2 \times 10^{-\mathbf{38}}$
Largest Value	$\approx 6.6 \times 10^{\mathbf{4}}$	$\approx 3.4 \times 10^{\mathbf{38}}$
Precision		
Next Representable > 1	$1 + 2^{-\mathbf{10}} \approx 1.000977$	$1 + 2^{-\mathbf{7}} \approx 1.007812$

In our investigation of the training–inference mismatch, we identify a surprisingly simple yet highly effective remedy that avoids complex algorithmic or engineering fixes. Rather than introducing additional machinery, we focus on a more fundamental factor: numerical precision. We find that merely switching the precision from the now-dominant BF16 format (Dean et al., 2012; Kalamkar et al., 2019) to the earlier Float16 (FP16) format (Micikevicius et al., 2017) substantially mitigates the policy mismatch and improves training stability. This section revisits the history and characteristics of these floating-point formats to shed light on this counterintuitive but powerful result.

3.1 FP16 vs. BF16

Floating-point formats represent real numbers by two components: exponent bits, which determine the range (how large or small a value can be), and mantissa bits (also known as fraction bits), which determine the precision (how finely values can be distinguished within that range). Both FP16 and BF16 use 16 bits in total, but they allocate these bits differently, resulting in distinct trade-offs between range and precision (see Table 1).

FP16 (IEEE 754 half-precision) allocates 5 bits to the exponent and 10 bits to the mantissa. The relatively large mantissa gives FP16 higher numerical precision, allowing it to represent small differences between nearby values accurately. However, its limited 5-bit exponent severely constrains the dynamic range, making FP16 prone to overflow (values exceeding the representable maximum) and underflow (values rounding to zero). Training with FP16 often requires stability techniques such as loss scaling to mitigate these issues (see Section 3.2).

BF16 (bfloat16), introduced by Google, allocates 8 bits to the exponent—matching the range of the 32-bit FP32 format—and only 7 bits to the mantissa. This design provides a wide dynamic range comparable to FP32, making BF16 highly resistant to overflow and underflow, at the cost of reduced precision. The resulting numerical robustness under low precision is the key reason for its widespread adoption in large-scale deep learning systems.

3.2 Stabilizing FP16 Training with Loss Scaling

The primary challenge with FP16's limited range is gradient underflow, which can be effectively solved early in the history of mixed-precision training with **loss scaling** (Micikevicius et al., 2017). The procedure is straightforward:

1. The loss is multiplied by a large scaling factor S before backpropagation.
2. This scales up all gradients by S, shifting small gradient values out of the underflow region and into the representable range of FP16, thus preserving them.
3. Before updating the weights, the gradients are scaled back by dividing S.

Modern implementations have further improved this with dynamic loss scaling. The scaling factor S is automatically adjusted during training, increased if no overflows (infinity values in gradients) are detected for a number of steps, and decreased immediately if an overflow occurs.

Crucially, these loss scaling techniques are standard, mature components in mainstream training frameworks (e.g., PyTorch (Paszke et al., 2019), Megatron (Shoeybi et al., 2019)). Enabling them typically requires **only a single configuration change or a few lines of code**, making the adoption of FP16 training both simple and robust (see Appendix B for a thorough discussion on managing the overflow and underflow in FP16).

3.3 The Rise of BF16 in Modern LLM Training

While effective, dynamic loss scaling complicates distributed training because it necessitates a synchronization before optimizer step to check overflows and align the scaling factor across workers.

The introduction of BF16 on hardware like Google TPUs and later NVIDIA GPUs (starting with the Ampere architecture) is a game-changer. Having the same dynamic range as FP32, BF16 offered a "drop-in" replacement for FP32 that obviates meticulous loss scaling. Its resilience to overflow and underflow made training LLMs more stable and straightforward. Consequently, BF16 quickly became the de-facto standard for modern mixed-precision training.

3.4 Why FP16 is the Key for RL Fine-Tuning

While BF16's stability is an advantage for pre-training models, our findings reveal that its low precision is the origin of the training-inference mismatch.

Modern RL frameworks often use different engines or optimized kernels for training and inference. Even if both are configured to use BF16, subtle differences in their implementation (e.g., CUDA kernel optimizations, parallel strategies) can lead to different rounding errors. When these small discrepancies accumulate over a sequence of tokens during autoregressive sampling, the resulting probability distributions for π and μ can diverge significantly. This divergence is the source of the biased gradients and the deployment gap discussed earlier.

This is precisely why switching to FP16 provides a fundamental solution. With its 10 mantissa bits, FP16 offers 8 times more precision (2^{10} values vs. 2^7 values) than BF16. This higher fidelity means that the outputs of the training and inference engines are much more likely to be numerically identical. The increased precision creates a buffer that absorbs the minor implementation differences between the two engines, reducing the accumulation of rounding errors and policy divergence.

For RL fine-tuning, the dynamic range of the model's weights and activations has already been established during pre-training. Therefore, the extreme range of BF16 is less critical, while the precision it sacrifices becomes a dominant drawback. By reverting to FP16, we trade the unnecessary range of BF16 for the critical precision, effectively closing the gap between training and inference without any complex algorithmic or engineering workaround.

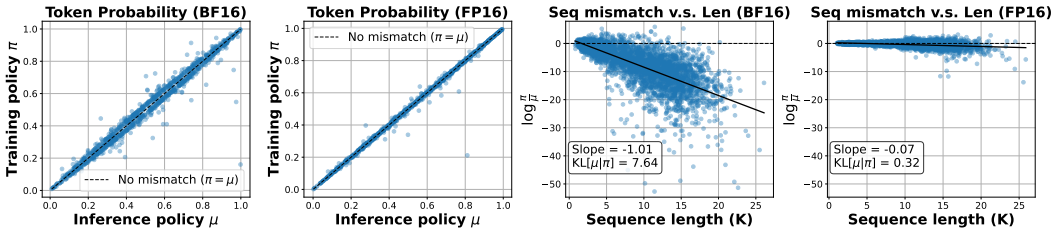

Figure 2: FP16 significantly reduces the training-inference mismatch. The left two plots show the token-level probability distribution, and the right two plots present the sequence-level log probability ratio between the inference (μ) and training policies (π). Dashed lines in black denote perfect precision without mismatch.

3.5 Offline Analysis Results

Before proceeding to RL fine-tuning, we first perform an offline analysis on the effects under different precisions. We generate 32 responses per question using temperature 1.0 and no top-p sampling (so that μ is directly comparable to π), and evaluate the token probabilities with the same model weights using DeepSpeed (Rasley et al., 2020) for training and vllm (Kwon et al., 2023) for inference, under both BF16 and FP16. The left two plots in Figure 2 show that FP16 notably reduces the mismatch between μ and π, with token probabilities more tightly concentrated around the perfect consistency. Beyond token-level discrepancies, we also analyze sequence-level mismatch, since $\frac{\pi(y|x)}{\mu(y|x)}$ serves as the unbiased importance sampling weight for a full response. The right two plots in Figure 2 depict the distribution of sequence-level log-probability ratios across different generation lengths. The results clearly indicate that BF16 introduces an **exponentially larger mismatch**, which is worsened with longer responses due to cumulative autoregressive errors, whereas FP16 maintains the mismatch at a much milder level (approximately 24$\times$ smaller).

4 A Sanity Test for RL Algorithms

To rigorously assess the reliability and robustness of RL algorithms, we introduce a novel sanity test. Standard benchmarks often contain a mix of problems with varying difficulty, including questions that are either overly trivial or unsolvable by the initial model. Trivial questions waste computational resources, while unsolvable ones make it difficult to determine whether poor performance stems from a flawed algorithm or the model's inherent limitations. Our sanity test is designed to remove this ambiguity with efficiency. By creating a perfectible dataset where every problem is known to be solvable but not trivial, we can cleanly isolate and evaluate an RL algorithm's ability to unlock a model's latent potential. On this perfectible dataset, **a reliable RL algorithm should theoretically be able to achieve 100% training accuracy**.

We construct this perfectible dataset by filtering out those overly trivial and unsolvable questions for the initial model. Specifically, we unroll 40 responses for each problem in the MATH dataset (Hendrycks et al., 2021), and only keep problems where the initial accuracy is between 20% and 80%. This process yielded a targeted dataset of 1,460 questions for the DeepSeek-R1-Distill-Qwen-1.5B model (Guo et al., 2025). The smaller size of this dataset makes achieving near-100% accuracy computationally feasible, allowing for efficient and conclusive testing.

We define our sanity test with a clear criterion: an RL algorithm passes if its training accuracy converges above a high threshold (e.g., 95%). An algorithm that fails this test can be considered unreliable or fundamentally flawed, as it is unable to guide the model to solve problems known to be within its reach. While passing is not a guarantee of universal success, **failing is a strong indicator of an ill-suited algorithm design**, making this test a crucial diagnostic tool.

Under this sanity test, we evaluate several representative RL algorithms, particularly those designed to address the training-inference mismatch (see Section 2.1). These include a vanilla GRPO baseline (specifically, the Dr.GRPO variant from Equation (3)) (Shao et al., 2024; Liu et al., 2025c); two corrected GRPO versions using a token-level TIS (Equation (4)) from Yao et al. (2025) and a sequence-level MIS (Equation (5)) from Liu et al. (2025a); and the standard policy gradient algorithm with importance sampling (Equation (2)). In addition, we include GSPO (Zheng et al., 2025b) in our experiments, although it was primarily designed to address the mismatch introduced by MoE models.

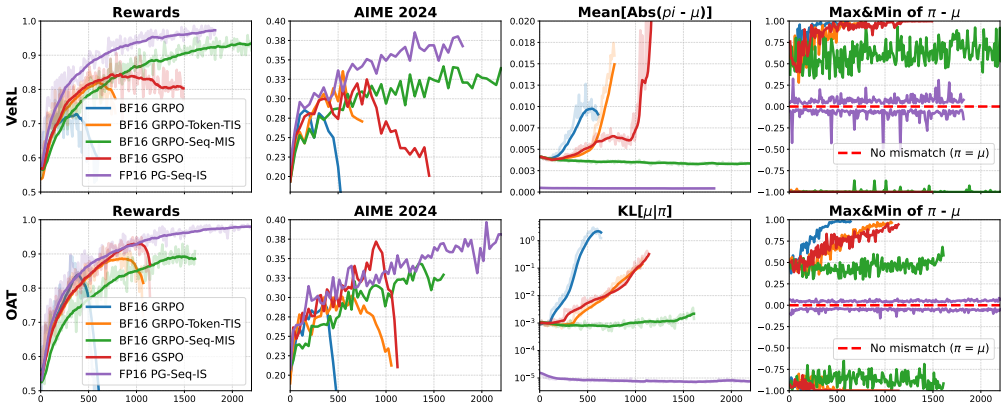

Figure 3: Simply switching from BF16 to FP16 stabilizes and prolongs RL training. The basic importance-weighted policy gradient in FP16 outperforms all baselines in BF16.

4.1 Comparison with Existing Algorithmic Corrections

To ensure robustness and rule out implementation-specific artifacts, we conducted experiments across two different frameworks: VeRL (Sheng et al., 2024) and Oat (Liu et al., 2025b). The results, shown in Figure 3, highlight the instability of existing methods when using BF16 precision.

The vanilla GRPO baseline collapses early in training, reaching a peak accuracy of only 73% in VeRL and 84% in Oat before its performance degrades. The token-level TIS correction (Yao et al., 2025) prolongs training slightly but ultimately fails, collapsing after reaching 82% (VeRL) and 88% (Oat) accuracy, an observation that aligns with findings from Liu et al. (2025a). Surprisingly, GSPO demonstrates more stable training for a longer period than GRPO with token-level TIS, achieving higher rewards despite not using the inference policy μ at all.

Among all the algorithmic corrections in BF16, only GRPO with sequence-level MIS (Liu et al., 2025a) maintains stable training without collapsing. However, this stability is costly. The method suffers from slow convergence due to the high variance of its sequence-level importance ratio (see Figure 2). Even at its peak, it exhibits a significant performance gap compared to our FP16 approach. It achieves a maximum training accuracy of only 95% (vs. 99% in FP16) and a score of 34% (vs. 39% in FP16) on the AIME 2024 benchmark, demonstrating a clear performance ceiling.

The Efficacy of FP16 Precision In contrast to these algorithmic approaches, simply switching both training and inference precision from BF16 to FP16 provides a dramatic improvement. As shown in Figures 1 and 11, the FP16 training runs are significantly more stable, converge much faster, and achieve substantially higher final rewards and evaluation scores across all tested algorithms. This result demonstrates that addressing the mismatch at the precision level is a more direct and effective solution than applying unstable or inefficient algorithmic corrections.

The most surprising finding is that FP16 fundamentally improves the behavior of importance sampling. The sequence-level ratio, which is notoriously high-variance, becomes much more concentrated and stable in FP16 (see Figure 2). This stabilization **makes it practical to use the unbiased policy gradient estimator without any modifications** (Equation (2)). As shown in Figure 3, this simple, unbiased approach, when powered by FP16, dramatically outperforms all algorithmic corrections in BF16.

4.2 Reviewing RL Algorithms under FP16

We then reviewed the performance of various RL algorithms when trained with FP16 precision. As shown in Figure 4, the performance differences between algorithms become almost indistinguishable (see Figure 7 for full results). We attribute this convergence in performance to the significantly reduced training-inference mismatch in FP16, which effectively transforms the optimization problem into a nearly on-policy setting. In this state, the complex corrections offered by different algorithms provide little to no additional benefit.

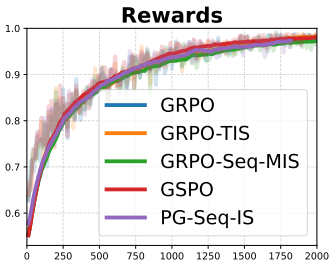

Figure 4: Comparisons between various algorithms on FP16.

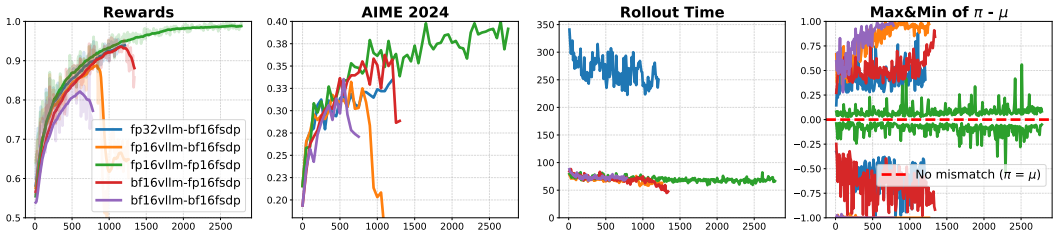

Figure 5: Ablation on the precision combinations.

4.3 Ablation on Precision Combinations

To isolate the effects of training and inference precision, we conducted an ablation study, using vLLM (Kwon et al., 2023) for inference and PyTorch FSDP (Zhao et al., 2023) for training. The results are presented in Figure 5.

When training with BF16 precision, we found that increasing the inference precision consistently prolonged training stability and improved performance. Notably, when paired with FP32 inference, the training run became fully stable with no signs of collapse. However, this stability came at an immense cost: FP32 inference was nearly three times slower than FP16 or BF16 inference, making this combination impractical for large-scale experiments.

In contrast, training with FP16 precision proved far more effective. Even with a mismatched BF16 inference, FP16 training was more stable and achieved better downstream performance. This highlights that high-precision training is the critical factor, likely because it provides less noisy and more accurate gradients for policy optimization.

Unsurprisingly, using FP16 for both training and inference yielded the best results. It not only produced the lowest mismatch but also resulted in the most stable training dynamics. It successfully reached nearly 100% training accuracy in the sanity test without any loss of speed, demonstrating a clear superiority in both stability and efficiency.

5 Broader Evaluation

Having demonstrated FP16's effectiveness on our controlled sanity test (Section 4 and Appendix E), we now validate these findings across more diverse settings, including MoE RL, alternative model families, and a wider range of tasks and frameworks.

5.1 Model Generalization

Mixture of Experts. MoE RL training is known to be unstable and often requires sophisticated stabilization strategies (Zheng et al., 2025b; Ma et al., 2025). Distinct parallelization strategies and precision-sensitive top-k expert selections can amplify the training–inference mismatch compared to dense models. We conduct RL experiments on Qwen3-30B-A3B-Base (Yang et al., 2025) with three algorithms: GRPO-Seq-MIS, GRPO-Token-TIS, and PG-Seq-TIS (see Appendix F.3 for details). FP16 exhibits greater stability and consistently higher training rewards (see (i), (j), (k) in Figure 1) as well as higher validation accuracies (see (i), (j), (k) in Figure 11). The improvement holds across all three algorithms, confirming that FP16 effectively improves the training stability and enhances overall performance of MoE models.

Large Dense Models. We experiment with Qwen3-14B-Base following the DAPO recipe (Yu et al., 2025) (see Appendix F.4 for details). As shown in Figure 1 (l), FP16 training rewards increase substantially faster than those with BF16. Figure 11 (l) further demonstrates higher validation accuracy on AIME 2024, highlighting the potential of FP16 for large-scale RL training.

Other Model Families. To rule out Qwen-specific effects, we train OctoThinker-3B (Wang et al., 2025b), a model mid-trained from Llama3.2-3B (Grattafiori et al., 2024) on reasoning-intensive data, using GRPO on the standard MATH dataset. As shown in Figure 1 (g), BF16 training destabilizes after around 150 steps, while FP16 continues to train smoothly without collapse.

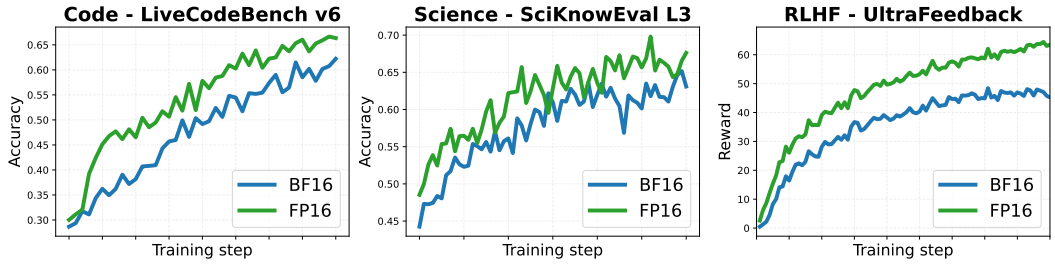

Figure 6: Comparisons between BF16 and FP16 on code (RLVR), science (RLVR), and RLHF tasks. Megatron and SGLang serve as training and inference engines for code and science tasks; FSDP and vLLM are used for the RLHF task.

5.2 Task Generalization

Code. We fine-tune Qwen3-8B (Yang et al., 2025) on the LCBv6 subset of LiveCodeBench (Jain et al., 2024), which contains 131 contest-style coding problems. Following Hübotter et al. (2026); Samadi et al. (2025), we split test cases into public and private unit tests, using public tests for training and private tests for validation. Validation accuracy is reported in the left plot of Figure 6.

Science. We fine-tune Qwen3-8B (Yang et al., 2025) on the L3 reasoning subsets from SciKnowEval (Feng et al., 2024), covering undergraduate-level questions in Chemistry, Physics, Biology, and Materials Science. Validation accuracy is reported in the middle plot of Figure 6.

RLHF. We fine-tune Qwen3-4B-Instruct-2507 (Yang et al., 2025) on the UltraFeedback dataset (Cui et al., 2023) using Skywork-Reward-Llama-3.1-8B (Liu et al., 2024) as the reward model, with training reward reported in the right plot of Figure 6. We additionally evaluate the step-150 checkpoints on AlpacaEval 2.0 (Dubois et al., 2024) to assess downstream quality. Results in Table 6 show higher alignment scores for FP16 than BF16 in this evaluation, providing evidence that the observed benefits extend to the RLHF setting.

LoRA. LoRA (Hu et al., 2022) has recently regained popularity in LLM RL (Wang et al., 2025a; Schulman & Thinking Machines Lab, 2025) due to its efficiency and performance comparable to full fine-tuning. We train Qwen2.5-Math-1.5B (Yang et al., 2024) on the standard MATH dataset using GRPO-Token-TIS (Equation (4)), applying LoRA to all layers with rank 32 and scaling factor $\alpha = 64$. Following Schulman & Thinking Machines Lab (2025), we adopt a slightly larger learning rate (4×10^{-5}) than that used in full fine-tuning. As shown in Figure 1 (h), BF16-based LoRA training collapses after roughly 600 steps, whereas FP16 maintains stable training throughout.

5.3 Framework Generalization

To confirm that FP16's gains are not tied to a specific software stack, we validate across multiple training and inference engines. In addition to our default setup of FSDP and vLLM, we also cover Megatron (Shoeybi et al., 2019) (MoE, code, and science experiments) and DeepSpeed (Rasley et al., 2020) (via the Oat framework in sanity test) as training engines, and SGLang (Zheng et al., 2024) (code and science experiments) as an inference engine. Across all combinations, FP16 consistently exhibits much lower training–inference mismatch (see Figure 12) and better performance than BF16, confirming that its effectiveness is framework-agnostic.

6 Conclusion

This work demonstrates that the training-inference mismatch, a major source of instability in RL fine-tuning, is fundamentally a problem of numerical precision. While existing algorithmic fixes are often complex and inefficient, we show that simply switching from the standard BF16 format to the higher-precision FP16 format can virtually eliminate the mismatch. This single, efficient change leads to more stable training, faster convergence, and superior performance, proving that addressing the problem at the precision level is a more effective strategy. We conclude that FP16 should be reconsidered as a foundational option for robust RL fine-tuning of LLMs.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation. We have not used generative AI tools to generate synthetic data sets, design or provide feedback on research methodology or experiments, implement methods, clean and reformat dataset, support qualitative and thematic data analysis, interpret results, and help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, draft parts of a research paper, edit a research paper to improve readability, suggest a structure for a research paper, propose a title or keywords for a research paper. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, et al. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*, 2025.

Zhoujun Cheng, Shibo Hao, Tianyang Liu, Fan Zhou, Yutao Xie, Feng Yao, Yuexin Bian, Yonghao Zhuang, Nilabjo Dey, Yuheng Zha, Yi Gu, Kun Zhou, Yuqi Wang, Yuan Li, Richard Fan, Jianshu She, Chengqian Gao, Abulhair Saparov, Haonan Li, Taylor W. Killian, Mikhail Yurochkin, Zhengzhong Liu, Eric P. Xing, and Zhiting Hu. Revisiting reinforcement learning for llm reasoning from a cross-domain perspective, 2025. URL https://arxiv.org/abs/2506.14965.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, et al. Ultrafeedback: Boosting language models with scaled ai feedback. *arXiv preprint arXiv:2310.01377*, 2023.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, et al. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*, 2025.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Kehua Feng, Xinyi Shen, Weijie Wang, Xiang Zhuang, Yuqi Tang, Qiang Zhang, and Keyan Ding. Sciknoweval: Evaluating multi-level scientific knowledge of large language models. *arXiv preprint arXiv:2406.09098*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Horace He. Defeating nondeterminism in llm inference. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250910. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, Siyuan Li, Liang Zeng, Tianwen Wei, Cheng Cheng, Bo An, Yang Liu, and Yahui Zhou. Skywork open reasoner series, 2025. Notion Blog.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, et al. Reinforcement learning via self-distillation. *arXiv preprint arXiv:2601.20802*, 2026.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13:9, 2024.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. *arXiv preprint arXiv:2410.18451*, 2024.

Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, Qian Liu, and Yu Shen. When speed kills stability: Demystifying rl collapse from the inference-training mismatch, 2025a. https://yingru.notion.site/When-Speed-Kills-Stability-Demystifying-RL-Collapse-from-the-Inference-Training-Mismatch-271211a558b7808d8b12d403fd15edda.

Zichen Liu, Changyu Chen, Chao Du, Wee Sun Lee, and Min Lin. Oat: A research-friendly framework for llm online alignment. `https://github.com/sail-sg/oat`, 2025b.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025c.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. `https://github.com/agentica-project/deepscaler`, 2025.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing moe reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble pipeline parallelism. *arXiv preprint arXiv:2401.10241*, 2023.

Penghui Qi, Zichen Liu, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Optimizing anytime reasoning via budget relative policy optimization. *arXiv preprint arXiv:2505.13438*, 2025.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

Mehrzad Samadi, Aleksander Ficek, Sean Narenthiran, Siddhartha Jain, Wasi Uddin Ahmad, Somshubra Majumdar, Vahid Noroozi, and Boris Ginsburg. Scaling test-time compute to achieve ioi gold medal with open-weight models. *arXiv preprint arXiv:2510.14232*, 2025.

John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Ling Team, Bin Han, Caizhi Tang, Chen Liang, Donghao Zhang, Fan Yuan, Feng Zhu, Jie Gao, Jingyu Hu, Longfei Li, Meng Li, Mingyang Zhang, Peijie Jiang, Peng Jiao, Qian Zhao, Qingyuan Yang, Wenbo Shen, Xinxing Yang, Yalin Zhang, Yankun Ren, Yao Zhao, Yibo Cao, Yixuan Sun, Yue Zhang, Yuchen Fang, Zibin Lin, Zixuan Cheng, and Jun Zhou. Every attention matters: An efficient hybrid architecture for long-context reasoning. *arXiv preprint arXiv:2510.19338*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Shangshang Wang, Julian Asilis, Ömer Faruk Akgül, Enes Burak Bilgin, Ollie Liu, and Willie Neiswanger. Tina: Tiny reasoning models via lora. *arXiv preprint arXiv:2504.15777*, 2025a.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025b.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Feng Yao, Liyuan Liu, Dinghuai Zhang, Chengyu Dong, Jingbo Shang, and Jianfeng Gao. Your efficient rl framework secretly brings you off-policy rl training, August 2025. https://fengyao.notion.site/off-policy-rl.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Chujie Zheng, Kai Dang, Bowen Yu, Mingze Li, Huiqiang Jiang, Junrong Lin, Yuqiong Liu, An Yang, Jingren Zhou, and Junyang Lin. Stabilizing reinforcement learning with llms: Formulation and practices. *arXiv preprint arXiv:2512.01374*, 2025a.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025b.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024.

A Discussions

Rethinking the Precision Tradeoff in RL Fine-Tuning Numerical precision is a foundational choice in the LLM training stack, yet this choice has long been dominated by BF16 for both pre-training and post-training, prized for its wide dynamic range and ease of use. Our results, however, suggest this default deserves careful rethinking for RL fine-tuning. In this phase, the training-inference mismatch becomes a critical source of instability, and BF16's low precision exacerbates this problem. We demonstrate that by simply trading BF16's wide dynamic range for FP16's higher precision, one can achieve significantly more stable RL training, faster convergence, and superior final performance.

It is important to note that we are not claiming FP16 is a universally optimal choice. The pursuit of efficiency may lead developers to even lower precisions like FP8. In LLM pre-training, for example, BF16 may remain preferable for its wider dynamic range and more stable training dynamics. Ultimately, we hope this work inspires the community to reconsider FP16 as a powerful and often more suitable alternative for stabilizing RL fine-tuning.

The Bias-Variance Tradeoff under BF16 Precision Our results in Section 4.1 reveal a bias-variance trade-off among RL algorithms operating under BF16 precision. Methods with lower variance but higher bias (like GRPO, token-level TIS, and GSPO) initially converge quickly but prove unstable and eventually collapse. Conversely, less biased algorithms that more accurately correct for the policy mismatch (like PG-Seq-IS and GRPO-Seq-MIS) achieve stability but at the cost of high variance, which slows their convergence.

This trade-off, however, becomes far less critical under FP16 precision. By fundamentally reducing the training-inference mismatch, FP16 naturally lowers both the bias induced by the mismatch and the variance of the importance sampling corrections. This enhanced stability allows even the most naive policy gradient estimator to converge efficiently, creating a training dynamic where all tested algorithms perform well and the tension between stability and speed is effectively resolved.

B Managing Overflow and Underflow in FP16

FP16 has a smaller numerical range than BF16. Values that are too large in magnitude can overflow, while very small values can underflow and round to zero. We describe how standard training tools address these issues, then report the numerical stability observed in our training and inference experiments.

B.1 Safeguards for FP16 Training

In the backward pass, underflow can round small gradients to zero and remove their contribution to learning. Dynamic loss scaling addresses this problem, as described in Section 3.2. It scales up the loss before backpropagation to preserve small gradients, then scales the gradients back before updating the weights. If NaN or Inf values appear in the gradients, the training framework skips the update and lowers the scale. This prevents invalid gradients from changing the weights and reduces the risk of further gradient overflow.

In the forward pass, underflow rounds very small activations to zero, which usually has little effect on the model's output. The main concern is therefore overflow. Loss scaling does not change forward activations, so separate safeguards are needed. Normalization layers such as LayerNorm (Ba et al., 2016) help control activation magnitudes. Automatic mixed precision (`torch.autocast`) uses FP32 for sensitive operations such as softmax and FP16 for expensive matrix multiplications. Tensor rescaling can also improve numerical stability, as in Megatron's `apply_query_key_layer_scaling` option. Overflow that survives is not fatal. The training framework detects the resulting NaN or Inf and skips the affected optimization step. Training fails only when overflow persists across many consecutive steps, which is atypical: activations large enough to overflow repeatedly indicate a model that is already diverging and would perform poorly under any precision.

B.2 Evidence from Training Runs

Table 2: Logged gradient-norm events in FP16 runs. "Steps" is the number of logged training steps; NaN/Inf indicates an overflow event, either from forward or backward pass.

Run	Steps	NaN	Inf
MoE GRPO-Seq-MIS	250	0	10
MoE GRPO-Token-TIS	193	1	2
MoE PG-Seq-TIS	250	0	6
Dense 14B	230	4	5
Sanity PG-Seq-IS	1,823	0	9
Sanity PG-Seq-MIS	2,500	2	8
Sanity PG-GSPO	2,189	0	14
Sanity GRPO	2,031	1	6
Sanity GRPO-Token-TIS	2,500	0	14
Sanity GRPO-Seq-MIS	2,478	2	7

All our FP16 runs used dynamic loss scaling without manual tuning for individual runs, and no run failed to complete because of a numerical error. Table 2 reports how often the logged gradient norm was NaN or Inf, where we intentionally turned off the `torch.autocast`. These values can result from overflow in either the forward or backward pass. Each run recorded between 3 and 14 such events over 193 to 2,500 logged steps. These observations show that occasional numerical errors were manageable in our tested settings.

Training in FP16 at scale also has precedent. InstructGPT, OpenAI's 175B model (Ouyang et al., 2022), explicitly states that all models use FP16 weights and activations. The released Llama 2 models (Touvron et al., 2023) were likewise trained in FP16, as the precision type recorded in their official repository indicates.

B.3 Inference with a 235B Model

To check forward-pass stability in a model larger than those in our training experiments, we evaluated Qwen3-235B-A22B using vLLM with a 32K context (temperature 1.0, top-$k = -1$, top-$p = 1.0$). FP16 inference completed without numerical failures and achieved scores close to BF16 on AIME 2025 and AIME 2026 (Table 3). This result supports the use of FP16 for inference of a large model.

Table 3: Inference-only evaluation of Qwen3-235B-A22B under FP16 and BF16. Mean@16 and Pass@16 are reported for AIME 2025 and AIME 2026.

	AIME 2025			AIME 2026		
Precision	Mean@16	Pass@16	Length	Mean@16	Pass@16	Length
FP16	0.7854	0.9092	15,332	0.8250	0.9215	14,298
BF16	0.7750	0.8762	14,972	0.8438	0.9294	14,185

C Additional Background

C.1 Variance Reduction of Importance Sampling

While theoretically sound, the estimator of Equation (2) often suffers from high variance, particularly in the context of LLMs where responses are long, leading to extreme probability ratios. To mitigate this, techniques that trade a small amount of bias for a significant reduction in variance, such as Truncated Importance Sampling (TIS) (Espeholt et al., 2018) and Masked Importance Sampling (MIS) (Liu et al., 2025a), have been proposed:

$$\nabla_\theta \mathcal{J}_{\text{pg-tis}}(x) = \mathbb{E}_{y\sim\mu(\cdot|x,\theta')}\left[\min\left(\rho, C\right) \cdot \nabla_\theta \log \pi(y|x,\theta) \cdot A_y\right], \tag{6}$$

$$\nabla_\theta \mathcal{J}_{\text{pg-mis}}(x) = \mathbb{E}_{y\sim\mu(\cdot|x,\theta')}\left[\rho \cdot \mathbb{I}\left\{\rho \le C\right\} \cdot \nabla_\theta \log \pi(y|x,\theta) \cdot A_y\right], \tag{7}$$

where C is a clipping hyperparameter and $\mathbb{I}\{\cdot\}$ is the indicator function. These methods stabilize training by controlling the magnitude of the importance weights.

The equations above write the sequence-level analogues used for the policy-gradient comparison. In particular, the token-level TIS correction from Yao et al. (2025) is defined in the main text by applying a ratio at each token, whereas the sequence-level form above is included to make the variance-reduction comparison explicit.

D Deployment Precision Evaluation

We begin by sampling 32 responses per question from the AMC and AIME benchmarks (Li et al., 2024), using the DeepSeek-R1-Distill-Qwen-1.5B model (Guo et al., 2025) with recommended decoding settings (temperature 0.6 and top-p 0.95) under various precisions. Each entry in Table 4 is the mean task score from these 32 samples. The scores are largely comparable, suggesting that higher inference precision alone does not necessarily yield improvements.

This is further validated by repeating this evaluation on a fine-tuned model under FP16. To assess the deployment flexibility of an FP16-trained model, we evaluated its performance under different inference precisions. We took the final DeepSeek-R1-Distill-Qwen-1.5B model from our sanity test (Section 4.1, Figure 3), which was fine-tuned using the vanilla unbiased importance-weighted policy gradient (Equation (2)) in FP16. As shown in Table 5, the model's performance when deployed in BF16 is comparable to its performance in FP16. This result aligns with our earlier findings in Table 4 and demonstrates that a model trained with FP16 precision can indeed be deployed under BF16 without a significant performance drop, confirming its practical utility.

We hypothesize that this phenomenon is due to the discrete, auto-regressive nature of token generation, which is inherently robust to minor numerical inaccuracies. Mistakes introduced at one step can be effectively corrected or rendered irrelevant by the model's subsequent token choices.

Table 4: Mean evaluation scores of DeepSeek-R1-Distill-Qwen-1.5B over 32 sampled responses per problem under different inference precisions and token budgets.

dtype	AMC23		AIME24	
	8K	32K	8K	32K
BF16	50.38	62.35	22.60	29.90
FP16	50.60	63.10	20.10	30.94
FP32	51.54	62.42	22.30	28.44

Table 5: Mean evaluation scores of an FP16-trained model over 32 sampled responses per problem, with 32K maximum response length.

precision	AMC23	AIME24
FP16	74.85	40.21
BF16	74.55	42.60

E Additional Details on Sanity Test

In our sanity test, all experiments use DeepSeek-R1-Distill-Qwen-1.5B as the initial model, with a context length of 8,000. We run each experiment on 8 NVIDIA A100 80G GPUs (see Appendix E.3 for hardware generalization). For each policy iteration (Schulman et al., 2017), we use a batch size of 64 questions (with 8 rollouts per question) and perform 4 gradient steps. For algorithms in the GRPO family, we set the `clip_higher` to 0.28 by default (Yu et al., 2025). The clipping threshold for importance sampling methods (Equation (4) and Equation (5)) is set to $C = 3$.

E.1 Training Dynamics

Our experimental results in Figure 3 reveal an interesting phenomenon: algorithms that eventually collapse consistently exhibit a growing training-inference mismatch beforehand, making it a potential early-warning signal (see Figure 3). During this period, the policy difference $\pi(\cdot|\theta') - \mu(\cdot|\theta')$ also converges to extreme values, where one policy's probability approaches 1 while the other's approaches 0, despite using the same copy of weights. We suspect this is driven by a particular optimization bias, though further validation is required. In contrast, stable algorithms maintain a bounded mismatch. Crucially, FP16 training shows a much lower mismatch level than any BF16 method. This inherent stability at the precision level explains why a simple policy gradient with FP16 can outperform all existing, more sophisticated solutions.

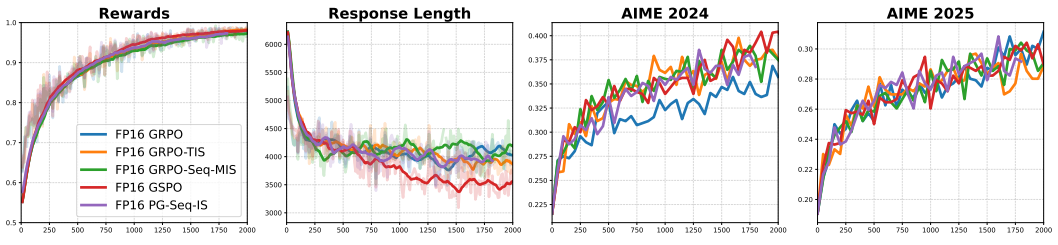

Figure 7: The full evaluation results for FP16 comparisons.

E.2 Additional Results of FP16 Comparisons

We put the full results of FP16 comparisons in Figure 7. We observed the original GRPO scored slightly lower on the AIME 2024 benchmark; however, it also scored slightly higher on AIME 2025, making it difficult to draw a definitive conclusion about its relative performance.

E.3 Hardware Generalization

To ensure our findings are not an artifact of a specific hardware architecture, we conducted an ablation study comparing NVIDIA A100 and H100 GPUs. While most of our experiments in sanity test (Section 4) were performed on A100s, we replicated the GRPO-Token-TIS experiment on H100. As shown in Figure 8, the resulting curves are largely similar. This strong similarity indicates that our conclusions regarding precision and stability are not specific to a single GPU type and are likely to generalize across different hardware platforms.

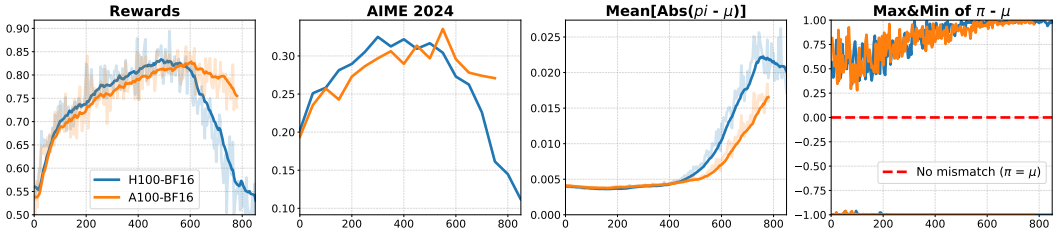

Figure 8: Comparison between A100 and H100 hardware with GRPO-Token-TIS algorithm.

E.4 Multi-seed Runs

We run FP16 PG-Seq-IS and BF16 GRPO under the sanity-test setting with two seeds per method, as shown in Figure 9. For FP16, the two training curves largely overlap. For BF16 GRPO, the collapse occurs at similar steps under both seeds. Both observations indicate that the training dynamics are robust to the random seed.

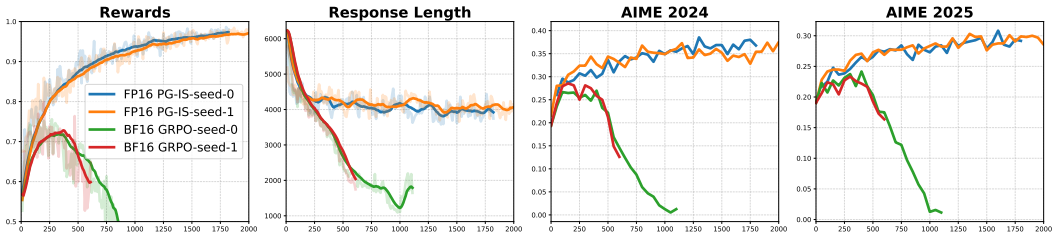

Figure 9: Multi-seed runs. FP16 remains stable across seeds, whereas BF16 collapses at similar steps.

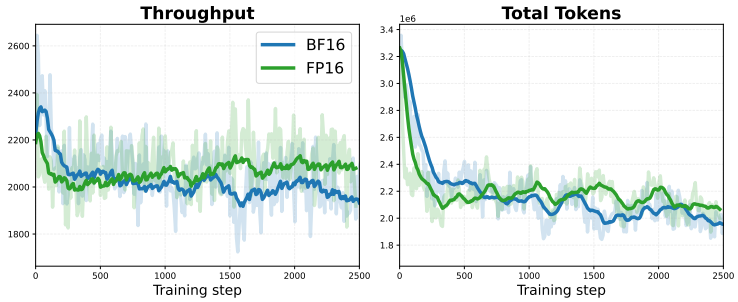

Figure 10: Logged end-to-end throughput comparison for BF16 and FP16.

E.5 End-to-end Throughput

We measure the end-to-end throughput of GRPO-Seq-MIS in the VeRL setup. As shown in Figure 10, FP16 reaches the same throughput as BF16, so switching precision introduces no noticeable overhead.

F Additional Details on Broader Evaluation

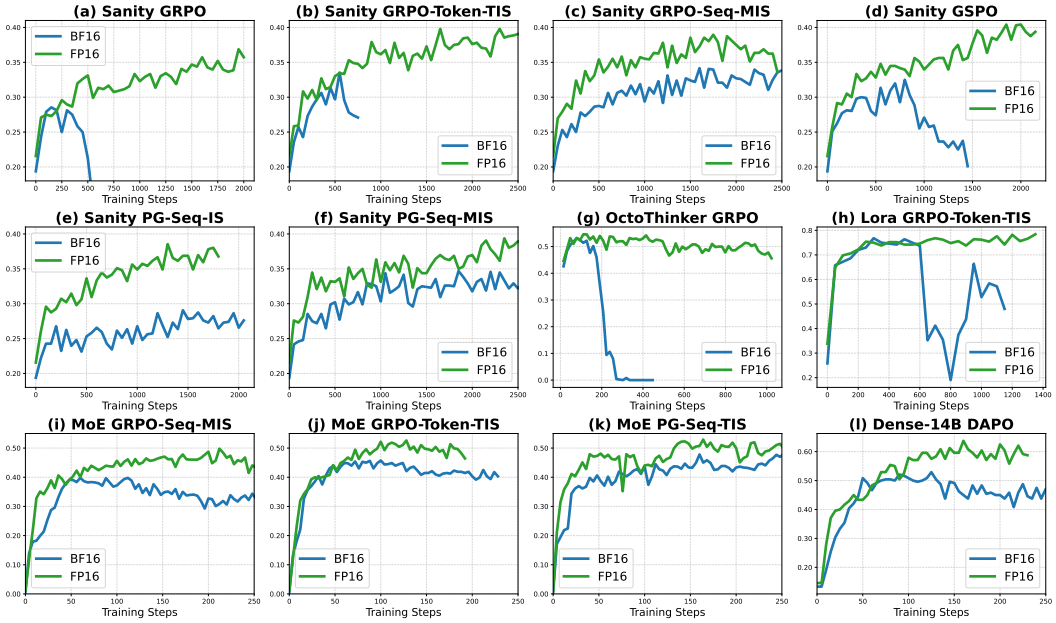

Figure 11: Evaluation comparisons on AIME2024 between BF16 and FP16 across various frameworks, algorithms, datasets and training regimes.

While Figure 1 presents the training reward curves under different precisions, Figure 11 shows evaluation results using checkpoints trained with these precisions. The results indicate that FP16-trained models generalize well to unseen benchmarks, further supporting our claim.

F.1 Additional Results on Framework Generalization

Figure 12 shows lower training–inference mismatch for FP16 than BF16 across various tasks and engines, confirming that its effectiveness is framework-agnostic.

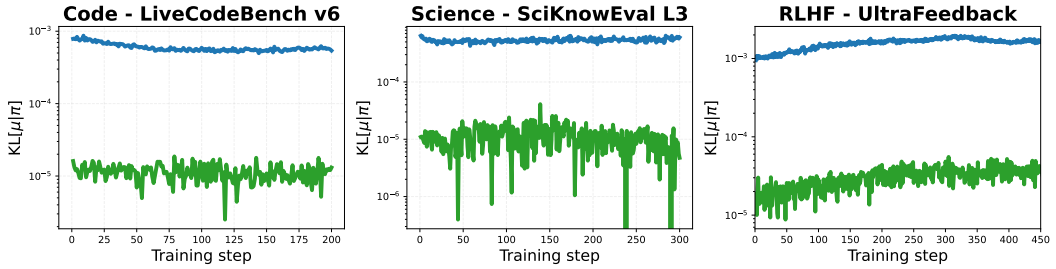

Figure 12: KL divergence comparisons between BF16 and FP16 on code, science, and RLHF tasks. Megatron and SGLang serve as training and inference engines for code and science tasks; FSDP and vLLM are used for the RLHF task.

F.2 RLHF Evaluation

We additionally evaluate the step-150 checkpoints on AlpacaEval 2.0 (Dubois et al., 2024) to assess downstream quality. Results in Table 6 show higher alignment scores for FP16 than BF16 in this evaluation, providing evidence that the observed benefits extend to the RLHF setting.

Table 6: AlpacaEval 2.0 results for RLHF fine-tuning with BF16 and FP16.

Model	LC Win Rate	Win Rate	Avg. Length
Initial model	59.39	69.38	3147
+ BF16 GRPO fine-tuning	77.05	72.20	1756
+ FP16 GRPO fine-tuning	**78.63**	**75.81**	1778

Table 7: Hyperparameters used for RL training of MoE models and large dense models.

Parameter	**MoE RL**	**Large dense RL**
`trainer.nnodes`	8	8
`trainer.n_gpu_per_node`	8	8
`model.path`	Qwen3-30B-A3B-Base	Qwen3-14B-Base
`vllm_version`	0.10.0	0.10.0
`data.train_batch_size`	512	512
`data.gen_batch_size`	N/A	1536
`data.max_prompt_length`	2048	2048
`data.max_response_length`	20480	20480
`rollout.n`	16	16
`rollout.temperature`	1.0	1.0
`rollout.top_p`	1.0	1.0
`val_kwargs.temperature`	0.6	1.0
`val_kwargs.top_p`	1.0	0.7
`actor.ppo_mini_batch_size`	32	32
`actor.ppo_max_token_len_per_gpu`	22528	22528
`optim.lr`	1e-6	1e-6
`optim.lr_warmup_steps`	N/A	10
`optim.weight_decay`	0.0	0.1
`optim.betas`	[0.9, 0.95]	[0.9, 0.999]
`optim.eps`	1e-15	1e-8
`algorithm.use_kl_in_reward`	False	False
`actor.use_kl_loss`	False	False
`actor.clip_ratio_high`	0.28	0.28
`actor.clip_ratio_low`	0.2	0.2
`actor.clip_ratio_c`	N/A	10.0
C in Equations (6) and (7)	3.0	N/A
`actor.loss_agg_mode`	seq-mean-token-sum-norm	token-mean
`overlong_buffer.enable`	False	True
`overlong_buffer.len`	N/A	4096
`overlong_buffer.penalty_factor`	N/A	1.0
`filter_groups.enable`	False	True
`filter_groups.metric`	N/A	acc
`filter_groups.max_num_gen_batches`	N/A	10

F.3 MoE RL

As for experiments of MoE RL, we use Qwen3-30B-A3B-Base as the base model. The training data comes from DAPO-Math-17k (Yu et al., 2025), and we conduct online evaluation on AIME 2024 using the `avg@32` metric. The training is performed with the VeRL framework (Sheng et al., 2024), and the key hyperparameters are summarized in Table 7.

Dr.GRPO (Liu et al., 2025c) proposes using a constant normalizer instead of a token-count-based normalizer. Notably, the open-source VeRL implementation does not correctly implement this. We refer to corrected version as "seq-mean-token-sum-norm" for `actor.loss_agg_mode` in VeRL.

F.4 RL ON LARGE DENSE MODELS

For experiments on large dense models, we use Qwen3-14B-Base as our base model. The training data is sourced from the mathematical domain dataset curated by Cheng et al. (2025). They aggregated recent math reasoning collections including OR1 (He et al., 2025), DAPO (Yu et al., 2025), and DeepScaler (Luo et al., 2025), and then performed deduplication and filtering to derive a final collection of 54.4k math training samples. We conduct online evaluation on AIME 2024 using the `avg@8` metric. The training algorithms and hyperparameters follow the setup described in Yu et al. (2025), as summarized in Table 7.

G WHY FP16 REDUCES TRAINING-INFERENCE MISMATCH

This section formalizes the argument in Section 3: FP16's higher mantissa precision reduces rounding noise, the root cause of the training–inference mismatch.

G.1 FLOATING-POINT PRECISION AND ROUNDING ERROR

Both BF16 and FP16 represent a real number x using sign, exponent, and mantissa bits:

$$\hat{x} = \mathrm{Q}(x) = (-1)^s \cdot 2^e \cdot (1+f),$$

where f is the fractional part encoded in m mantissa bits. The two formats differ in how they allocate bits between range and precision:

Format	Mantissa bits m	Relative rounding error $\varepsilon_m = 2^{-m}$
BF16	7	$\approx 7.8 \times 10^{-3}$
FP16	10	$\approx 9.8 \times 10^{-4}$

The relative rounding error ε_m bounds the error incurred when representing any real x within the normal representable floating-point range:

$$|\mathrm{Q}(x) - x| \le \varepsilon_m \cdot |x|. \tag{8}$$

FP16 provides an $8\times$ tighter bound than BF16, since $2^{10}/2^7 = 8$.

G.2 PER-OPERATION ERROR PROPAGATION

Rounding errors compound through arithmetic. Let $\hat{a} = a(1+\delta_a)$ and $\hat{b} = b(1+\delta_b)$ be floating-point approximations of a and b, with $|\delta_a|, |\delta_b| \le \varepsilon_m$.

Addition. Two distinct sources of error contribute to $\widehat{a+b} \triangleq \mathrm{Q}(\hat{a}+\hat{b})$: errors inherited from the rounded inputs and the rounding introduced by the addition itself. Applying the triangle inequality:

$$|\widehat{a+b} - (a+b)| \;\le\; \underbrace{|(\hat{a}+\hat{b}) - (a+b)|}_{\text{(i) inherited}} \;+\; \underbrace{|\mathrm{Q}(\hat{a}+\hat{b}) - (\hat{a}+\hat{b})|}_{\text{(ii) operation rounding}}. \tag{9}$$

For term (i): $|(\hat{a}+\hat{b}) - (a+b)| = |a\delta_a + b\delta_b| \le (|a|+|b|)\varepsilon_m$. For term (ii): applying Equation (8) to $x = \hat{a}+\hat{b}$ and using $|\hat{a}|+|\hat{b}| \le (|a|+|b|)(1+\varepsilon_m)$:

$$|\mathrm{Q}(\hat{a}+\hat{b}) - (\hat{a}+\hat{b})| \;\le\; \varepsilon_m(|\hat{a}|+|\hat{b}|) \;\le\; (|a|+|b|)(1+\varepsilon_m)\varepsilon_m.$$

Combining:

$$|\widehat{a+b} - (a+b)| \;\le\; (|a|+|b|)\,(2\varepsilon_m + \varepsilon_m^2). \tag{10}$$

Multiplication. Since $\widehat{ab} = \mathrm{Q}(\hat{a}\hat{b})$, the rounding bound gives $\widehat{ab} = \hat{a}\hat{b}(1+\delta_{\mathrm{op}})$ with $|\delta_{\mathrm{op}}| \le \varepsilon_m$. Substituting $\hat{a} = a(1+\delta_a)$ and $\hat{b} = b(1+\delta_b)$:

$$\frac{\widehat{ab}}{ab} = (1+\delta_a)(1+\delta_b)(1+\delta_{\mathrm{op}}) = 1 + \underbrace{\delta_a + \delta_b + \delta_{\mathrm{op}}}_{\text{first-order terms}} + O(\varepsilon_m^2).$$

Taking absolute values:

$$\left|\frac{\widehat{ab}}{ab} - 1\right| \;\leq\; |\delta_a| + |\delta_b| + |\delta_{\text{op}}| + O(\varepsilon_m^2) \;\leq\; 3\varepsilon_m + O(\varepsilon_m^2). \tag{11}$$

In both cases, the dominant error is linear in ε_m, preserving the $8\times$ precision advantage of FP16 per operation.

G.3 SUMMARY

By Equations (10) and (11), the per-operation rounding error is linear in ε_m for both addition and multiplication. Since FP16's ε_m is $8\times$ smaller than BF16's, each operation accumulates $8\times$ less rounding noise. A token's log-probability is computed through many such operations, and smaller per-operation errors therefore translate to a smaller discrepancy between training and inference computations.